IDIAP COMMUNICATION REPORT

WHO WANTS TO BE A MILLIONAIRE?

Huseyn Gasimov[a] Aleksei Triastcyn[a] Petr Motlicek
Hervé Bourlard

Idiap-Com-03-2012

JULY 2012

[a]EPFL

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

"Who Wants To Be A Millionaire?"

Real-Time Game Controlled by Automatic Speech Recognition and Text-To-Speech Systems

Semester project, EPFL

Huseyn Gasimov (huseyn.gasimov@epfl.ch)
Aleksei Triastcyn (aleksey.tryastsyn@epfl.ch)
Petr Motlicek (petr.motlicek@idiap.ch)
Hervé Bourlard (bourlard@idiap.ch)

Lausanne, June 2012

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

idiap
RESEARCH INSTITUTE

Contents

Introduction

Nowadays, the number of new speech recognition applications grows extremely fast. Automatic Speech Recognition (ASR) is widely used in modern mobile phones: for web search, dictation, making calls, notes and reminders. Further, it becomes increasingly integrated with personal computers. Largely used Mac OS X and Windows operating systems both have integrated ASR tools, although do not enable using all its power. Gaming environment is one of relatively new applications for speech recognition. It becomes very promising field, yet it is not widely used in computer and mobile games.

Our semester project is the development of the computer game "Who wants to be a millionaire?" with integrated speech recognition and text-to-speech synthesis. The game is implemented for two operating systems (Linux and Mac OS X), assumed to work close to real-time, and without any additional facilities (built-in microphone, Java platform, etc.).

This report is divided into 3 main parts. First, we will introduce the architecture of the application, briefly describing each component of the system. Detailed overview of a speech recognition component is in the second part. Finally, we will shortly talk about some problems, we faced during the project and make conclusions.

Application architecture

"Who wants to be a millionaire?" is a cross-platform application. It is implemented using Java platform [3], and runs on Java Virtual Machine [4], which allows the application to work in variety of operating systems. Theoretically it can work on Mac OS, Linux and Windows, but for the moment it is bounded by ASR module to work only on Mac OS X and Linux.

The application is built up of the following components:

- Database of questions and answers
- Business logic (dialog manager) and Interface
- Text-to-Speech component
- Speech recognition component (keyword spotting component)

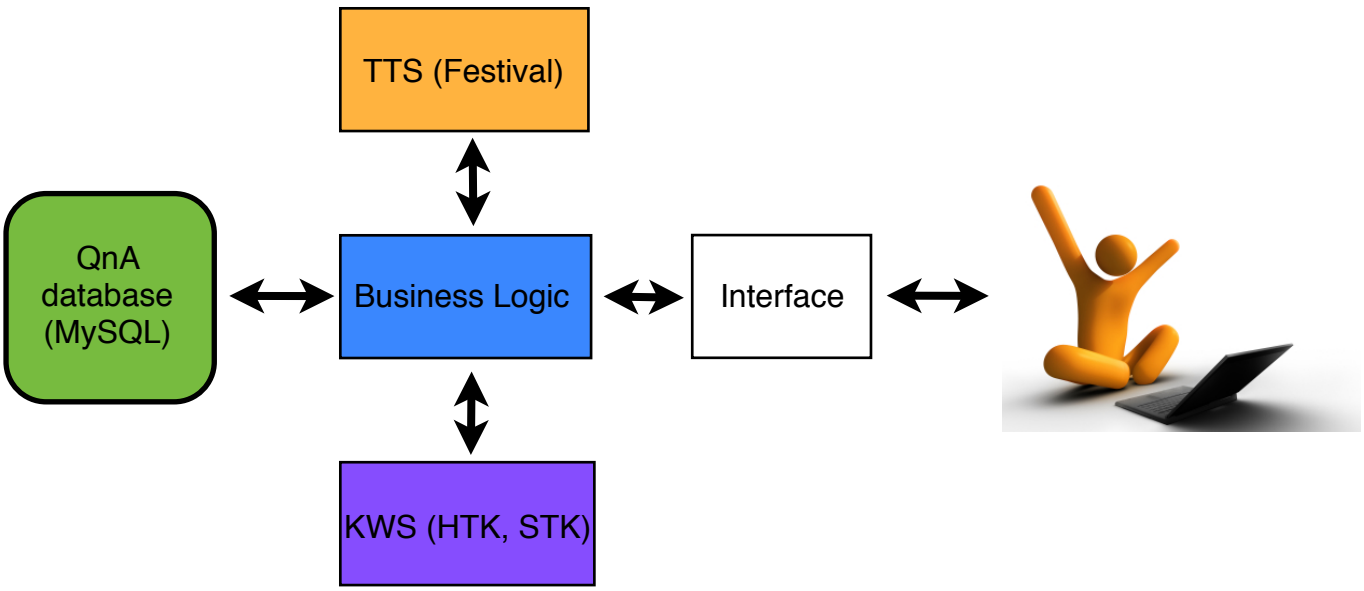

Fig. 1. Application architecture.

Currently "Millionaire" is developed as a non-distributed system. All its components are located on the same machine and connected through the file system (except for database connection, which is built over TCP connection). This solution is temporary, due to its inconvenience for users and sophisticated development of application for other platforms (including mobile phones).

Therefore, in the nearest future we plan to modify the application and switch to distributed architecture. On the client side only the user interface will be left, while all the other components will move on the server side.

Database

For storing a set of questions and answers (QnA) we use a relational database. This decision was made in order to maintain possible future extendibility of the QnA set and application as a whole. For the purposes of testing this database was filled with a small number of questions and answers, but for its further enlarging one can use a part of user interface dedicated to this goal.

To maintain the database we use the open source relational database management system - MySQL [5]. Creation of the database was done in MySQL Workbench, and later on it may be distributed with the application as an SQL-script.

The database consists of two tables, which are linked by foreign keys. Table TAnswers contains all possible answers for all the questions presented in the system. TQuestions keeps all the questions. For each question it stores links to the corresponding answers in TAnswers, the number of right answer and the difficulty of the question.

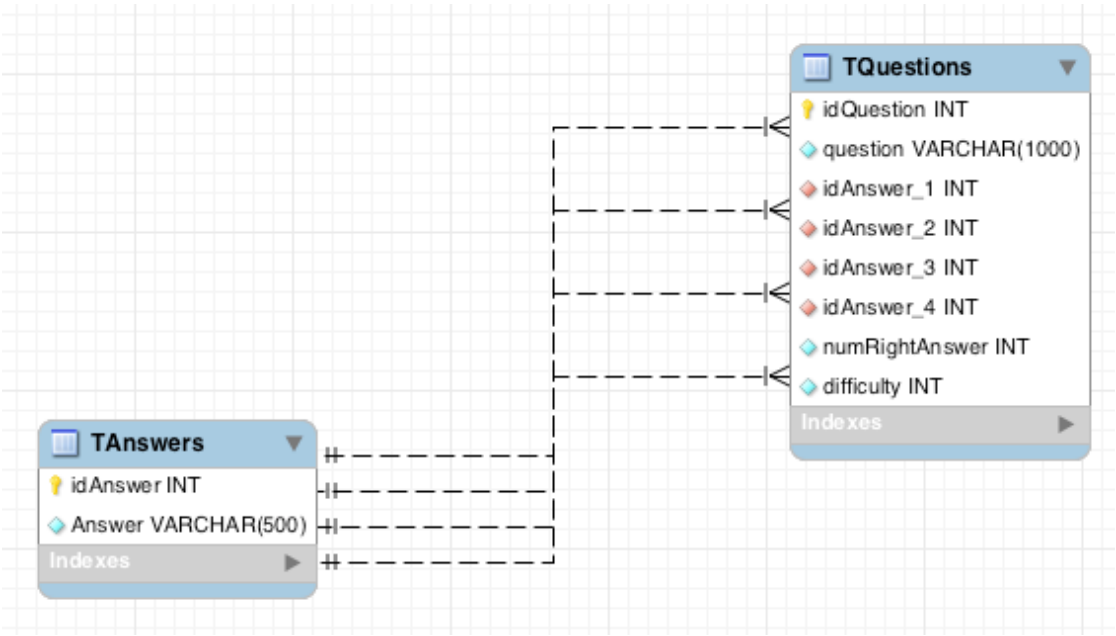

Fig. 2. Database scheme.

idQuestion	question	idAnswer_1	idAnswer_2	idAnswer_3	idAnswer_4	numRightAnswer	difficulty
1	What are you...	8	9	10	11	3	2
2	What might a...	12	13	14	15	3	2
4	How many wo...	4	5	6	7	4	1
7	In basketball,...	16	17	18	19	2	3
8	In what centur...	20	21	22	23	2	1
9	According to...	24	25	26	27	4	5
10	In which time...	28	29	30	31	1	4
11	Who invented...	32	33	34	35	4	6
12	According to...	36	37	38	39	3	10
13	The Latin wor...	40	41	42	43	2	11
14	The people w...	44	45	46	47	2	12
15	What did Ben...	48	49	50	51	2	13
16	What US state...	52	53	54	55	2	7
17	Mt. Goodwin...	56	57	58	59	3	14

Fig. 3. Data in TQuestions table.

Business Logic and Interface

The business logic and the interface parts were implemented, as mentioned above, in the Java programming language. The biggest advantage of using Java platform is that we can handle the compatibility of the application with different operating systems (OS). The range of OS working with Java is incredibly large, and all the restrictions in compatibility are caused by other parts of the software, such as ASR component.

Business logic (the functional algorithms that handle information exchange between a database, a user interface, and other components), which can also be called "dialog manager", is a central part of the application. It handles all the application interactions, including those between user interface and database, KWS or TTS components.

Interface is the only part of the system which interacts with user. It provides various controls and connects user to other components of the application. GUI includes several forms: Main, Game, AddQuestion and BestScores.

Fig. 4. Main form.

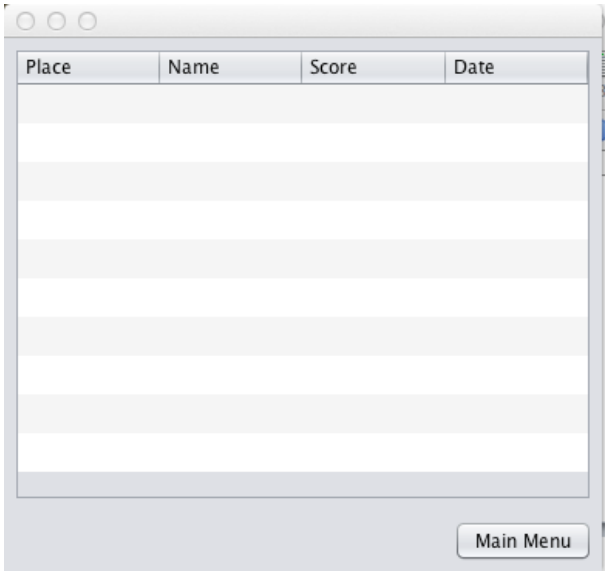

Fig. 5. BestScores form.

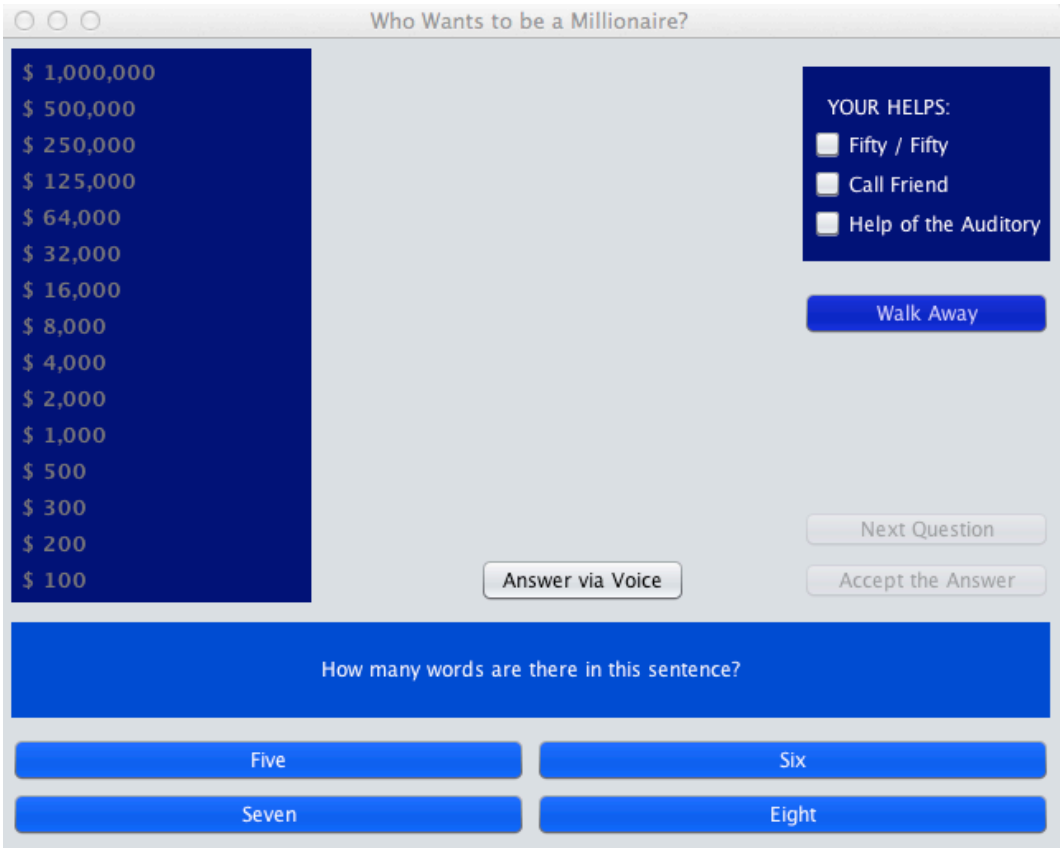

Fig. 6. Game form.

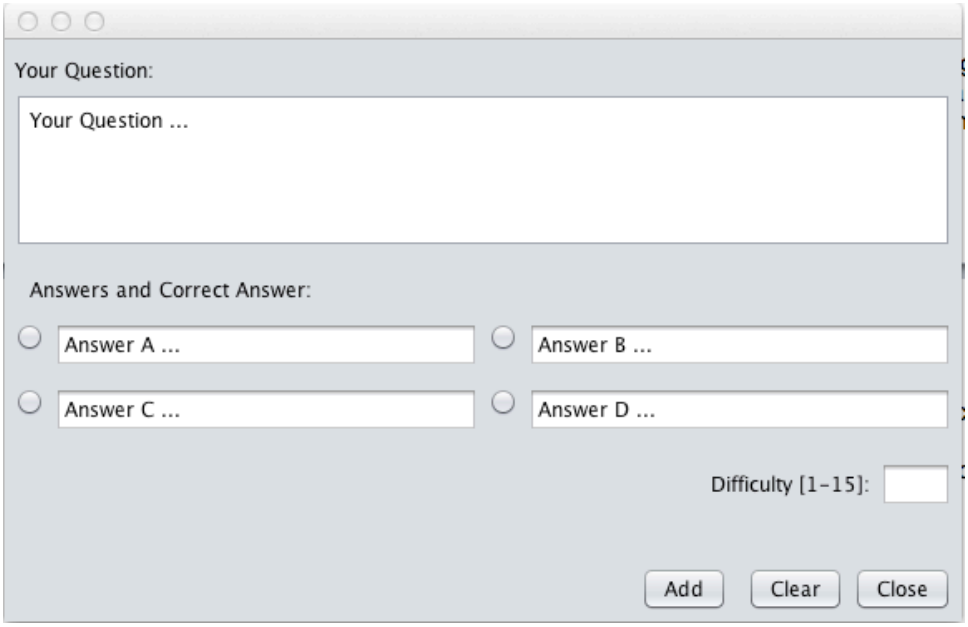

Fig. 7. AddQuestion form.

With the help of GUI users, beyond playing the game, can also check the best scores, or add new questions to the database. All the GUI forms are designed to be easily understandable by the user. The interface of the game process follows the appearance of the original talk-show "Who wants to be a millionaire?".

Text-to-Speech

In order to make the application more user-friendly, we have decided to incorporate a text-to-speech (TTS) block in it. Each question, prompted to user, is therefore pronounced by the TTS. The prompted speech is generated in real-time, by using an open source framework "Festival" [6].

Festival is a general framework for building speech synthesis systems. It offers a possibility to add a full text-to-speech support through the number of APIs. Festival's speech synthesis engine is based on Hidden Markov Models, which makes the generated speech properly coarticulated and smooth.

Keyword Spotting

Keyword spotting (KWS) component is the essential part of this application. In this part of the report we will give a detailed description of the keyword spotting process.

First of all, it is important to mention, that we did not implement a full speech recognition in the actual version of the software. Instead, a KWS block is used and for the current requirements it works reasonably well. The decision to substitute an ASR with KWS was stipulated by the need of using open vocabulary and dynamic words set for each question. In addition, since KWS is a detection technique, it is capable of providing word confidence scores. Due to that, the detected keywords can be accepted or rejected based on the obtained score.

The keyword spotting system is built on classical machine learning techniques. It employes Neural Networks (NN) and Hidden Markov Models (HMM). KWS module is divided into 3 parts (or stages of KWS process):

- feature extraction
- posteriors estimation
- decoding

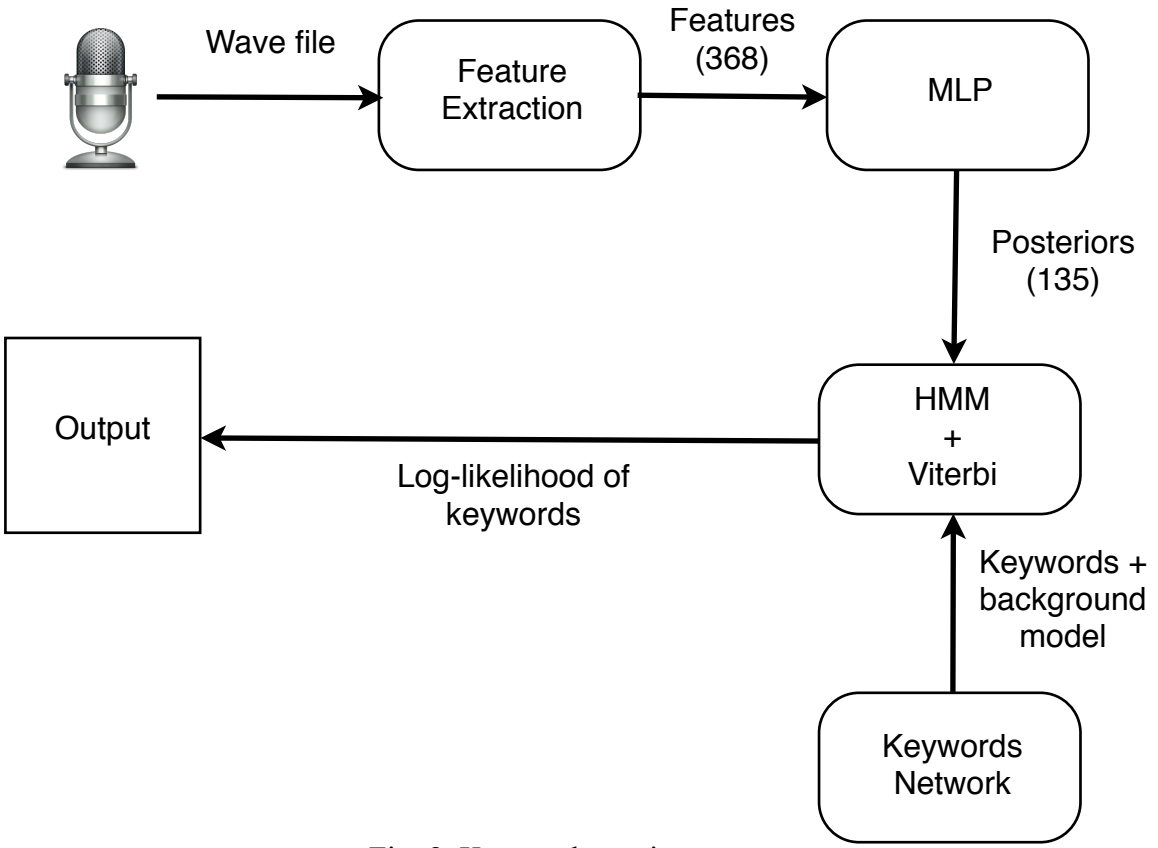

Fig. 8. Keyword spotting process.

Each of these stages is performed by a specific tool. The tools are taken from the open source speech recognition toolkits, HTK (Hidden Markov Model Toolkit) [7] and STK (HMM Toolkit STK from Speech@FIT) [8]. For feature extraction we use HCopy tool (from HTK), for posteriors estimation and decoding - SFeaCat and SLRatio (from STK) correspondingly.

Features extraction

Feature extraction is the first stage in our keyword spotting process. On this stage the input wave file is processed by the HCopy tool and the corresponding feature vectors are retrieved.

The input wave file must fulfill certain conditions: speech is sampled with 16 kHz and coded with 16 bit PCM.

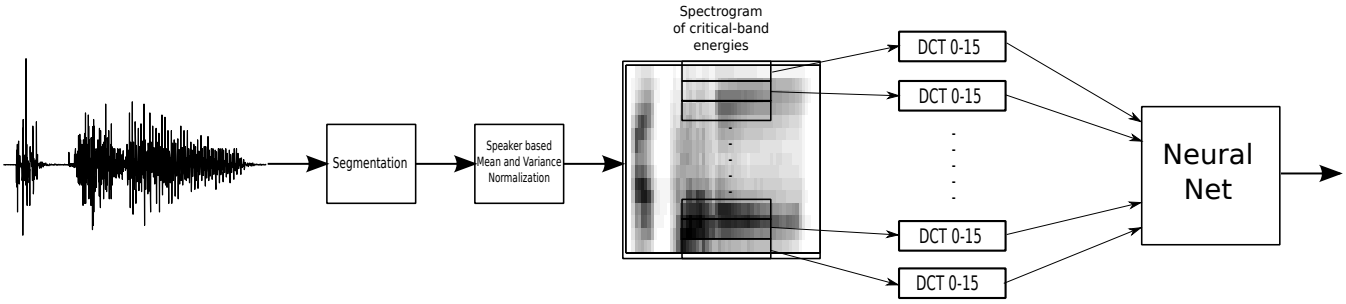

Fig. 9. Features extraction.

The scheme on Fig. 9 represents the whole process of features extraction. Segmentation and speaker normalization are currently not used in our application.

Features extraction algorithm performs the following steps:

1. Transform the input speech to frequency domain by applying the Fourier transform on speech segments generated using Hamming window of 25ms (frame rate 10ms).
2. Map the linear frequency bins into 23 critical bands (using mel scale) and obtain 23 critical-band-warped energies.
3. For each critical band apply DCT (discrete cosine transform) over the temporal context of 300ms.
4. Preserve only first 16 DCT coefficients (capturing the most of data variability).
5. Concatenate DCT coefficients obtained for each critical band into one vector of 368 components.

The result of the above algorithm – features vector estimated each 10 ms– is passed to the next tool for phones posteriors estimation.

Phone posterior estimation

Phone posteriors are estimated by a Neural Network, to be more precise - Multi-Layer perceptron (MLP) [9]. In our case, the MLP is trained to estimate the posterior probabilities of 45 phonemes (a standard set of english phonemes used in AMIDA project [10]). The training was performed on 150 hours of AMI and ICSI data. After training, MLP's parameters are stored in the file and loaded before each KWS run.

The MLP has a bottleneck architecture [1]. This structure implies containing, beside regular hidden layers, one or several narrow hidden layers, which allows NN to retrieve the most significant part of data, compressing the data this way.

More specifically, the MLP used in this project, is composed of 5 layers: input (368 units), output (135) and 3 hidden layers (4000, 30, 4000). The second hidden layer compresses the information and does not use non-linearity (its transfer function is linear as opposed to other layers).

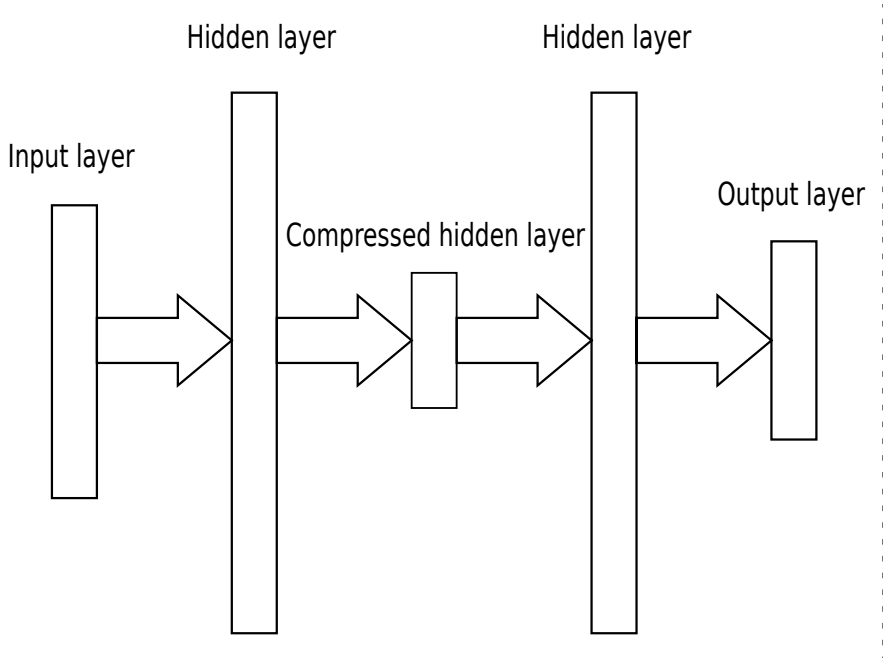

Fig. 10. Neural Network.

The output layer contains 135 neurons, and hence, the network outputs a vector of 135 components. These values represent state-dependent phone posteriors (i.e., 3 values per phoneme corresponding to HMM states).

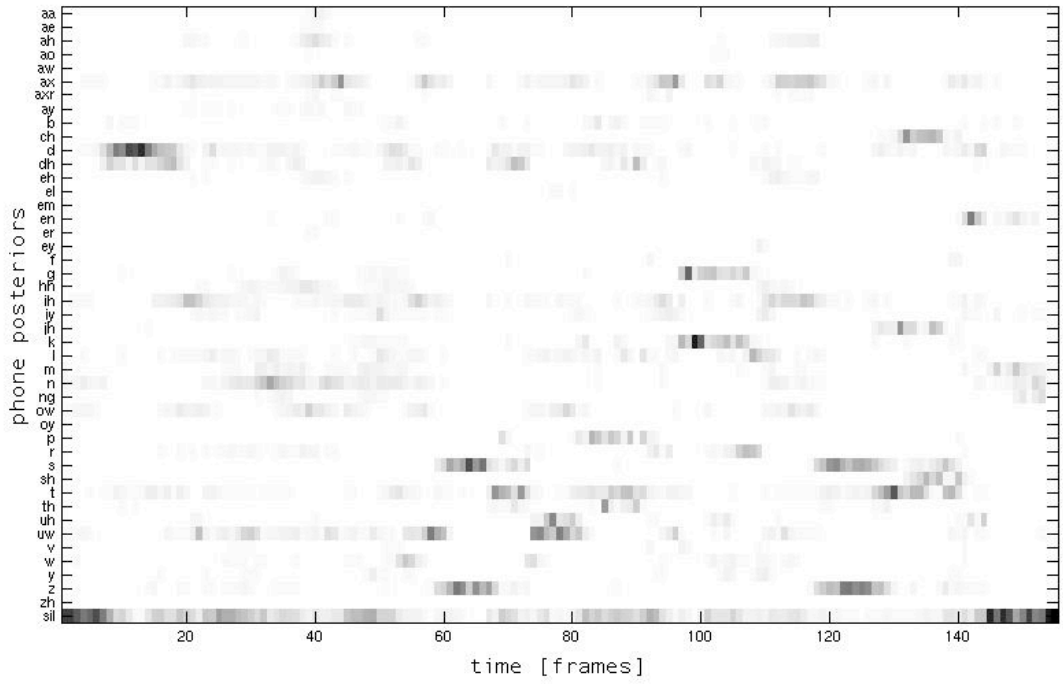

Fig. 11. Example of MLP output.

Decoding

Hidden Markov Models (HMMs) are used in the final step of keyword spotting process. Phone posterior estimates obtained by MLP are used in HMMs to represent emission probabilities. Then, the decoding network is created from a set of HMMs (representing each individual phone by 3 states) and Viterbi decoding algorithm is applied to derive the optimal path.

Keyword network's generation is an important point in out KWS setup. Keyword network is a graph model, which represents all the words that assumed to be detected. This is the key difference between concepts of keyword spotting and full automatic speech recognition. Unlike ASR, KWS restricts the decoding algorithm to consider only those words which are in the network.

The keyword network [2] contains keyword models, filler models and a background model. Keywords models are used to represent sought-for keywords. Background model allows estimating confidence scores for keywords, by computing log-likelihood ratio between paths given by keyword models and the background model. Filler models (or garbage models) are dedicated to model uninteresting parts of speech signal. Filler and background models share the same structure. They are built of HMMs of all the phonemes looped together.

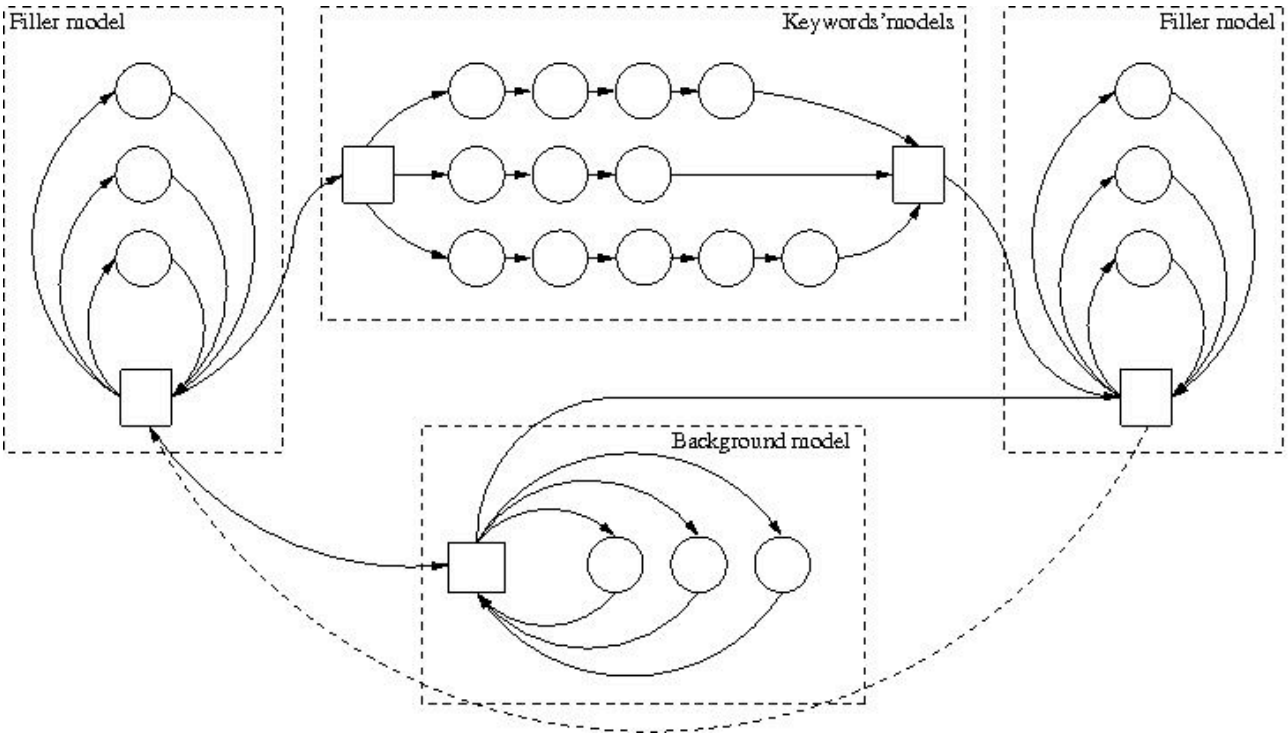

Fig. 12. Keywords network.

Decoding is performed by the Viterbi algorithm. This algorithm finds the most likely sequence of HMM states for each keyword model given the observations, and returns the log-likelihood for this sequence. In addition, it computes the log-likelihood for the background model, in order to estimate the log-likelihood ratios for each keyword, which will be taken as scores.

$$D = -2\ln\left(\frac{\text{likelihood for null model}}{\text{likelihood for alternative model}}\right)$$
$$= -2\ln(\text{likelihood for null model}) + 2\ln(\text{likelihood for alternative model})$$

"Null model" here refers to the current keyword model, while "alternative model" – to a background model.

In general, the decoding algorithm steps are:

1. Build keyword network.
2. Build HMM sequences for each keyword (by concatenating HMMs according to keyword network).
3. Estimate log-likelihoods for each keyword and background model by running the Viterbi algorithm.
4. Compute scores for each keyword (log-likelihood ratio).

Difficulties with Mac implementation

One of the main challenges of this project was porting the Linux speech tools on Mac OS X. The difficulties were caused by the differences between operating systems: OS X is a *Unix-based* system and Linux is a *Unix-like* system, and some OS components, used by third-party software, are different.

In general, building a software from its source code on Linux is much simpler. During the compilation it checks the dependencies, and if some additional library is needed, it will be automatically downloaded and installed. In the case of Mac we had to trace these dependencies by ourselves.

The main problems were faced during the installation of the STK tool. For building and installing the STK we had to find and install additional software, combine two different source distributions, modify the makefile, and even modify some of the source code files.

Conclusion

During this semester project we have created a voice-controlled application, running on Max OS X and Linux, which allows user to play the game "Who wants to be a millionaire?". This application uses HCopy, SFeaCat and SLRatio tools for KWS component and Festival TTS engine. It works close to real-time, with built-in microphone and could be fully operated by voice.

For the future we're planning to improve this application by adding a Voice Activity Detection (VAD), increasing the noise robustness, adding a block of speaker normalization, TTS interruption (barging) and analyzing the KWS output on a deeper level.

References

1. H. Bourlard, N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Boston: Kluwer Academic, 1994. Print.
2. P. Motlicek, F. Valente, I. Szoke. *Improving acoustic based keyword spotting using LVCSR lattices.* Proceedings of International Conference on Acoustic Speech and Signal Processing, Japan, 2012.
3. Java Platform website. URL: http://www.java.com/en/.
4. T. Lindholm, F. Yellin, G. Bracha, A. Buckley. *The Java™ Virtual Machine Specification. Java SE 7 Edition.* URL: http://docs.oracle.com/javase/specs/jvms/se7/html/index.html.
5. MySQL website. URL: http://www.mysql.com/.
6. The Festival Speech Synthesis System. URL: http://www.cstr.ed.ac.uk/projects/festival/.
7. Hidden Markov Model Toolkit. URL: http://htk.eng.cam.ac.uk/.
8. HMM Toolkit STK. URL: http://speech.fit.vutbr.cz/software/hmm-toolkit-stk.
9. M. Seeger. *Pattern Classification and Machine Learning* (Course notes). Probabilistic Machine Learning Laboratory, EPFL, 2012.
10. AMIDA Project webpage. URL: http://www.amiproject.org/.